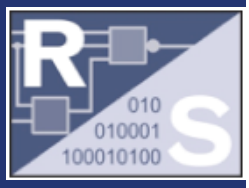
Rechnersysteme 2
Praktikum 1
SoC

Seminararbeit vorgelegt von Marcel Mann und Marcel Humm
Abgabe: 15. Januar 2018
Institut für Datentechnik | Fachgebiet Rechnersysteme | Prof. Dr.-Ing. Christian Hochberger

TECHNISCHE
UNIVERSITÄT
DARMSTADT



1 TODO

Kursiv schreiben, Funktionsnamen, Produktnamen, etc. Zeitformen, Gliederung: Aufgabenstellung Hardware (an sich vollständig) beta Protokolle (I2C SPI) beta Vorgehen:

Implementierung

Kommentare im Code anpassen an geänderte Signaturen

Port out in OLEDinit verschieben, testen ob geht SPI aufteilen nach spi und oled

Toolchain in Vorgehensweise: Jconfig-> vhdl/verilog? -> synthetisieren irgendwie auf Spartan hexen

UART genutzt über USB als Debug etc.

Herausforderungen: steile lernkurve Beispielcode OLED unzureichend kommentiert, schwer nachvollziehbar, bessere doku/anleitung wünschenswert, mit einfacheren Beispielen, unwesentliche Informationen zu viel

2 Aufgabenstellung

Im Rahmen dieses Praktikums sollte ein auf Ultraschall basierender Abstandsmesser mit Display entstehen. Dazu wurde jeder teilnehmenden Gruppe ein System-on-Chip Kit (SoC Kit) zur Verfügung gestellt. Dessen Basis bildet das SP601, ein FPGA-Board der Firma Xilinx, erweitert mit einem SRF02-Ultraschallsensor und einem OLED Display Modul der Firma Newhaven. Zur Entwicklung des Abstandsmessers sollte zunächst das Software-Tool JConfig zur Konfiguration der gegebenen Hardware verwendet werden. Darauf aufbauend galt es die Software-Treiber für die I^2C-Schnittstelle des Ultraschallsensors als auch die SPI-Schnittstelle des Displays zu implementieren. In einer Reihe von Zusatzaufgaben bestand weiterhin die Möglichkeit die notwendige Rechenleistung und Ausführungszeit der entwickelten Firmware durch die Verwendung von Interrupts und Code-Profiling zu optimieren.

3 Hardware

3.1 Ultraschallsensor

Beim SRF02 handelt es sich um einen kompakten Ultraschallsensor der Firma Devantech Ltd., der sich sowohl über einen I²C-Bus als auch über eine serielle RS232-Schnittstelle ansteuern lässt. Die verwendete Ultraschallfrequenz beträgt 40khz. Der messbare Bereich des Sensors liegt abhängig von Umgebung und Temperatur zwischen ca. 15cm und 6m. Eine einzelne Messung kann dabei bis zu 65ms dauern. Da der Sensor während eines Messvorgangs keine weiteren Befehle annehmen kann, liegt dessen maximale Messfrequenz somit bei 15 Messungen pro Sekunde. Standardmäßig kann der Sensor über die Adresse 0xE0 angesteuert werden. Über ein beschreibbares Befehlsregister kann ein Messvorgang mit Auswertung der gemessenen Distanz in Zentimetern, Zoll oder Mikrosekunden (Zeit bis Echo) gestartet werden. Das Ergebnis der Messung und die aktuell messbare Mindestdistanz können über vier weitere Register ausgelesen werden. Ein weiteres vorhandenes Register besitzt keine Funktion.

3.2 Display

Zur Ausgabe der vom Ultraschallsensor gemessenen Distanz das OLED-Display-Modul NHD-2.8-25664UCB2 mit einer Auflösung von 256 x 64 Bildpunkten der Firma Newhaven verwendet. Über den integrierten SSD1322-Controller von Solomon Systech kann das Display über eine serielle oder parallele Schnittstelle angesteuert werden. Für dieses Praktikum wird lediglich die serielle Schnittstelle (3-wire) mit einer Framebreite von 9 Bit verwendet.

3.3 FPGA-Board

Die Basis des verwendeten SoC-Kits bildet das SP601 Evaluation Kit der Firma Xilinx. Das um einen Spartan-6 FPGA aufgebaute Board bietet neben SPI- und I²C-Bussen auch Ethernet, USB zum Testen und Debuggen der Konfiguration mit JTAG und UART, diverse User-I/O-Pins, einen Oszillatoren zur Taktgenerierung, diverse Status-LEDs und 128 MB DDR2 RAM. Auf dem Spartan-6 FPGA wird der speziell für die Verwendung in FPGAs entwickelte 18 Bit SpartanMC-Softcore-Prozessor abgebildet.

4 Protokolle

4.1 UART

Universal Asynchronous Receiver Transmitter (UART) beschreibt eine serielle Schnittstelle zur asynchronen Übertragung über eine Datenleitung. Um eine asynchrone Kommunikation zu gewährleisten, einigen sich beide Kommunikationspartner außerhalb des Systems sowohl auf die Strukturierung der Daten als auch auf die verwendete Übertragungsrate. Ein Datenpaket besteht dabei immer aus einem Start-Bit (0), gefolgt von den zu übertragenden Daten und einem Stop-Bit (1). Im SpartanMC wird der Simple Universal Asynchronous Receiver Transmitter (UART Light) verwendet. Im Vergleich zu UART wird hier ein leichtgewichtigeres Interface verwendet und die Länge der Datenpakete auf 8 Bit beschränkt.

4.2 I²C

Bei I²C, kurz für Inter-Integrated Circuit, handelt es sich um einen seriellen multi-Master und multi-Slave fähigen Datenbus. Zwei Signalleitungen ermöglichen die Kommunikation zwischen jeweils zwei Komponenten nach dem Master-Slave-Prinzip. Über die Serial-Data-Line (SDA) können Master und Slave Informationen austauschen. Der dazu benötigte Takt wird vom Master über die Serial-Clock-Line (SCL) gegeben.

- **Verbindungsaufbau**
  Bei Inaktivität des I²C-Bus liegt an beiden Signalleitungen eine 1 an. In diesem Zustand kann ein angeschlossener Master eine Verbindung zu einem angeschlossenen Slave herstellen, indem er SDA auf 0 setzt. Nach diesem Startsignal kann mit jeder steigenden Flanke des SCL ein Bit übertragen werden. Ein Datenpaket besteht dabei aus jeweils 8 Bit.

- **Datenübertragung**
  Während der Datenübertragung zwischen Master und Slave darf sich das an SDA angeschlossene Signal nur ändern, während an SCL eine 0 anliegt. Das zunächst gesendete Datenpaket sollte die 7 Bit lange Adresse des zu adressierenden Slaves enthalten. Das achte Bit dieses Startpakets signalisiert die Lese- (1) oder Schreibabsicht (0) des Masters. Der Erhalt eines Datenpakets muss vom Empfänger mit einem Acknowledge-Bit bestätigt werden. Eine 0 signalisiert hierbei eine erfolgreiche Übertragung.

- **Verbindungstrennung**
  Nach Abschluss der Datenübertragung kann der I²C-Bus wieder in einen inaktiven Zustand gesetzt werden, indem SDA von 0 auf 1 geändert wird, während an SCL eine 1 anliegt.

4.3 SPI

Das Serial Peripheral Interface (SPI) beschreibt eine serielle vollduplex-fähige Kommunikationsschnittstelle über die sich ein einzelner Master mit einer Menge unterschiedlicher Slaves verbinden lässt. Alle angeschlossenen Komponenten teilen sich die drei Signalleitungen serial clock (SCLK), master out slave in (MOSI) und master in slave out (MISO). Zusätzlich existiert für jeden angeschlossenen Slave eine fest zugeordnete low-active slave select (SS) Leitung.

Eine Datenverbindung beginnt, indem der Master das SS-Signal des zu adressierenden Slaves auf 0 setzt und anschließend mit der Generierung eines Takt-Signals beginnt. Analog zu I²C wird auch hier mit jedem Takt ein Bit übertragen. Da jedoch bei SPI nur ein einzelner Master existiert, können Wortlänge, Taktfrequenz und Phasenlage (Datenübertragung bei steigender gegenüber fallender Taktflanke) von diesem an die jeweiligen Bedürfnisse unterschiedlicher Slaves angepasst werden. Den Wortanfang kann der ausgewählte Slave durch den vom Master gegebenen Takt erkennen.

5 Vorgehen

Um eine strukturierte Durchführung dieses Praktikums zu gewährleisten, haben wir uns zunächst entschieden die gegebene Aufgabenstellung in 6 aufeinander aufbauende Meilensteine zu gliedern.

5.1 Aktivieren des Ultraschallsensors

Da der Ultraschallsensor gemäß Aufgabenstellung über dessen I²C-Schnittstelle angesteuert werden sollte, galt es zunächst das I²C-Master-Modul des SpartanMC mit Hilfe von JConfig zu konfigurieren. Im nächsten Schritt wurde ein einfacher Treiber implementiert, der dem I²C-Master-Modul eine Reihe von Anweisungen übergibt. Um den Ultraschallsensor zu aktivieren, muss das I²C-Master-Modul zunächst gestartet und dabei ein zu verwendender Takt gegeben werden. Anschließend gilt es den Ultraschallsensor als Slave zu adressieren und in dessen Befehlsregister den Start-Befehl zur Entfernungsmessung in Zentimetern zu setzen. Die erfolgreiche Messung wird am Ultraschallsensor durch das Aufleuchten einer LED signalisiert.

5.2 Messwerte auslesen

Den begonnenen Treiber galt es im nächsten Schritt zu erweitern, um eine Einsicht der gemessenen Distanz des Ultraschallsensors zu ermöglichen. Da dieser gemäß seiner Spezifikation bis zu 65ms benötigt bis die Messwerte nach Beginn der Messung in den jeweiligen Registern abgelegt wurden, muss zunächst über einen sleep-Befehl entsprechend lange mit dem Auslesen gewartet werden. Anschließend gilt es den Ultraschallsensor erneut als Slave zu adressieren und diesem die mitzuteilen, welche Register der Master im nächsten Schritt auslesen möchte. Darauf folgt schließlich die Adressierung des Ultraschallsensors als Slave mit Lesebefehl. Nach der Übertragung der angegebenen Registerinhalte liegen diese in den Datenregistern des I²C-Master-Moduls ab und können vom Treiber über die UART-Schnittstelle auf der Konsole ausgegeben werden

5.3 Filtern

Die vom Ultraschallsensor ausgelesenen Messwerte können durch verschiedene Faktoren negativ beeinflusst werden. Neben statischen Fehlern wie der Verwendung einer falschen Berechnungsformel im Treiber zählen dazu auch dynamische Fehlerquellen basierend auf den physikalischen Eigenschaften der Ultraschallmessung. Um fehlerhafte Ausgaben zu vermeiden, wurde ein einfacher Ringspeicher implementiert. In diesem werden fortlaufend die letzten X Messergebnisse zwischengespeichert. Mit jedem eingehenden Ergebnis wird zunächst das älteste gespeicherte Ergebnis aus dem Ringspeicher verdrängt. Anschließend wird der aktuelle Inhalt des Ringspeichers kopiert, sortiert und dessen Median als aktuelle Distanz ausgegeben.

Weiterhin werden nur Werte in den Ringspeicher übernommen, die zwischen der kleinsten und größten messbaren Distanz des Ultraschallsensors liegen. Während es sich bei der größten messbaren Distanz um einen statischen Wert von 594cm handelt, ist die kleinste messbare Distanz abhängig von Umgebungsvariablen wie beispielsweise der Raumtemperatur. Beim Start der Programmausführung wird die aktuell kleinste messbare Distanz dementsprechend über die I²C-Schnittstelle vom Ultraschallsensor abgefragt und als Untergrenze für den Eintritt von Messwerten in den Ringspeicher übernommen.

5.4 Display ein- und ausschalten

Zur Ansteuerung des Displays über dessen SPI-Schnittstelle galt es zunächst das SPI-Master-Modul des SpartanMC mit Hilfe von JConfig zu konfigurieren. Anschließend muss das eigentliche SPI-Master-Modul aktiviert, dessen Wortbreite, Taktfrequenz und Phasenlage angepasst und das Display als Slave selektiert werden. Um das Display zu starten ist das Ausführen einer Initialisierungs-Sequenz notwendig. Dafür wird ein zusätzliches Port-Modul des SpartanMC benötigt, das über JConfig initialisiert und an den Reset-Eingang des Displays angeschlossen werden kann. Für die Implementierung des benötigten Display-Treibers konnte ein Großteil des vom Hersteller bereitgestellten Beispielscodes verwendet werden. Lediglich zwei Grundmethoden zum Schreiben von Befehlen und Daten galt es an unsere Konfiguration anzupassen. Nach Durchführung der vorherigen Schritte kann das Display nun durch die Initialisierungs-Sequenz und die anschließende Ausführung der im Beispielcode gegebenen init-Methode gestartet werden.

5.5 Ausgabe einfacher Strings auf dem Display

Im vom Hersteller bereitgestellten Beispielcode findet sich bereits eine ausführbare Methode zur Darstellung von Text auf dem Display. Deren Ausführung basiert dabei auf einer Abbildung der eingegebenen Textzeichen auf die zur Ausgabe benötigten ASCII-Werte. Neben der Einbindung dieser bereits gegebenen Struktur in den Display-Treiber, galt es einen Offset-Wert zu definieren, um die korrekte Zuordnung der ASCII-Werte zu gewährleisten. Die Ausgabeposition des eingegebenen Texts ist weiterhin abhängig von einem global im Display-Treiber definierten Shift-Wert und den der zur Ausgabe verwendeten Methode übergebenen Positionsparameter.

5.6 Ausgeben der Messwerte auf dem Display

Für den finalen Meilenstein dieser Abgabe galt es schließlich den gefilterten Integer-Wert in einen nullterminierten String zu übersetzen. Um nicht nur einen einfachen Zahlenwert auszugeben, haben wir uns weiterhin dafür entschieden diesem ein Prefix der Form „Distance: „ und ein Suffix „cm“ hinzuzufügen. Der fertige String kann nun mit der bereits konfigurierten Ausgabemethode des Display-Treibers auf dem Display ausgegeben werden.

6 Implementierung

Die Erläuterungen zu unserer Implementierung teilen wir in den Meilensteinen entsprechende Abschnitte ein. Hierbei gehen wir auf einige wichtige Funktionen ein und zeigen den entsprechenden Quellcode. Bei einfachen Hilfsfunktionen verzichten wir auf Quellcodeerläuterungen. Insgesamt besteht unsere Implementierung aus einer *main.c*-, *spi_driver.c*-, *i2c_driver.c*– und einer *functions.c*-Datei. Mit Ausnahme der main-Datei, wurde für jede C-Datei auch eine Header-Datei angelegt.

6.1 Aktivieren des Ultraschallsensors

Der erste Schritt bestand darin den I²C-Master einzurichten und darüber den Ultraschallsensor auszulesen. Der I²C-Master wird über seine externen Register angesprochen. Aktiviert wird er über das Setzen des Enable-Bits und das Festlegen des Prescalers in seinem Control-Register. Der Prescaler wird nach der Formel

$$Prescaler = peripheral\_clock/(5 * desired\_scl) - 1 = 60MHz/(5 * 40KHz) - 1 = 299$$

bestimmt. Für diesen Schritt, wurde die Methode $i2c_peri_enable$ implementiert.

```
void i2c_peri_enable(i2c_master_regs_t *i2c) {
        i2c -> ctrl = I2C_CTRL_EN | 299;
}
```

6.2 Messwerte auslesen

Ist der I²C-Master aktiviert, sind drei Schritte notwendig um einen Abstandswert zu erhalten:

- **Schritt 1: Anweisung zum Starten der Messung senden**
  Dazu wurde die Methode *i2c_peri_write* implementiert. Sie schreibt die Slave-Addresse, das Zielregister des Slaves und den zu sendenden Befehl in die Datenregister des I²C-Masters. Über dessen Befehlsregister wird nun die Übertragung gestartet.

  ```
  void i2c_peri_write(i2c_master_regs_t *i2c, unsigned int slave_address,
      unsigned int target_register, unsigned int command){
          i2c -> data[0] = slave_address;
          i2c -> data[1] = target_register;
          i2c -> data[2] = command;
          i2c -> cmd = I2C_CMD_STA | I2C_CMD_STO | (3 << 3);
          waitForTransfer(i2c);
  }
  ```

  Die *wait_for_transfer*-Methode überprüft mithilfe des Status-Registers des I²C-Masters, ob die Datenübertragung abgeschlossen ist. Erst dann, wir das Programm weiter ausgeführt. Um den Ultraschallsensor mit dem Start der Messung zu beauftragen, wird die Methode mit den entsprechenden Parametern aufgerufen.

- **Schritt 2: 65 Milisekunden warten** Um 65 Milisekunden zu warten bis die Messung sicher abgeschlossen ist, wurde eine erweiterte Sleep-Funktion implementiert. Die Dokumentation der in *sleep.h* implementierten *sleep*-Methode besagt, dass ein Aufruf derMethode mit dem Parameter n eine Verzögerung von $6 + 3 * n$ Takten auslöst. Bei einer Frequenz von 60MHz muss die Funktion also mit 19998 als Parameter aufgerufen werden. Dies erledigt die selbst implementierte $sleep_ms$-Methode, welche als Parameter die Anzahl der zu schlafenden Takte entgegennimmt.

- **Schritt 3: Die beiden Ergebnisregister auslesen** Um nun die gemessene Distanz auszulesen, wurde die Methode $srf02_read_dist$ implementiert. Sie teilt dem Slave mit, welche Register gelesen werden sollen, und liest diese anschließend aus. Da der Messwert über 256cm betragen kann, sind 2 Bytes notwendig um ihn darzustellen. Mit der einfachen Formel $High\_Byte * 256 + Low\_Byte$ wird der Wert berechnet und kann zurückgegeben werden. Über einen einfachen Aufruf der printf-Methode kann die gemessene Entfernung über die UART-Schnittstelle an den PC gesendet werden.

6.3 Filtern

Um Messwerte vor der Ausgabe zwischenspeichern zu können, haben wir einen Ringbuffer implementiert. Der Ringbuffer besteht aus einem Datenarray mit sieben Einträgen, den Pointern *head* und *tail*, welche die Anfangs- und Endpositionen der Daten im Array markieren und einem Zähler, welcher die Anzahl der aktuell im Array gespeicherten Werte festhält. Um den Buffer einfacher nutzen zu können, haben wir die Funktionen *put*, *get*, *isFull* und *isEmpty* implementiert.

Über die *b_put*-Methode landet jeder gemessene Wert im Buffer. Ist nach dem Einfügen des Wertes der Buffer voll, kann die weitere Verarbeitung stattfinden. Die *array_copy*-Methode kopiert nach jeder Messung den Inhalt des Buffers in ein zweites Array, welches anschließend mit einer einfachen Bubblesort-Implementierung sortiert wird. Ausgegeben wird nun der vierte Wert im sortierten Array, welcher den Median darstellt. Bei der gewählten Filtermethode sind mehrere Dinge zu beachten: Zu Programmstart benötigt es vier Messungen um den Ringbuffer mit Messwerten zu füllen, bevor ein gültiger Wert ausgegeben werden kann. Dies wird durch eine entsprechende Ausgabe angezeigt. Des weiteren benötigt auch eine Änderung der Distanz mindestens vier Messungen um zur Ausgabe zu gelangen. So lange dauert es nämlich, bis ein neuer Wert in die Mitte des Buffers gelangt. Da vier Messungen jedoch nur etwa 260ms benötigen, ist diese Einschränkung akzeptabel.

6.4 Display ein- und ausschalten

Zur Ansteuerung des Displays via SPI war ähnlich wie beim Ultraschallsensor zunächst das Konfigurieren des SPI-Masters nötig. Auch hier geschieht dies über das Setzen von Bits im Kontrollregister.

```
void spi_peri_enable(spi_master_regs_t *spi) {
        spi -> spi.spi_control |= SPI_MASTER_CTRL_EN;
        spi -> spi.spi_control &= ~SPI_MASTER_CTRL_BITCNT;
        spi -> spi.spi_control |= (9 << 8);
        spi -> spi.spi_control |= SPI_MASTER_CTRL_DIV;
        spi_peri_select(spi);
}
```

Ist der SPI-Master aktiviert und konfiguriert, muss das Disply über die Reset-Sequenz initialisiert werden. Um den gegebenen Treiber des Display verwenden zu können, musste die Methode *spi_peri_write* angepasst werden.

```
void spi_peri_write(spi_master_regs_t *spi, unsigned char data) {
        spi -> spi.spi_data_out = data;
        while (spi -> spi.spi_status & SPI_MASTER_STAT_FILL);
}
```

Damit ist das Display bereit zur Verwendung. Mithilfe der Befehle zum Ein- und Ausschalten der Displaybelechtung, konnte dies einfach überprüft werden.

6.5 Ausgabe einfacher Strings auf dem Display

Beim Testen der String-Ausgabe Methode fiel auf, dass die Zeichenketten nicht in der gewünschten Form auf dem Display erschienen. Nach kurzer Suche merkten wir, dass die in der *font.h*-Datei definierte Schriftart das Ausrufezeichen als erstes Zeichen gelistet hat. Allgemein werden Zeichen jedoch gemäß dem ASCII-Standard codiert, wobei das Ausrufezeichen erst Zeichen Nummer 33 ist. Um nun die richtigen Defniitionen zu erhalten, muss man vom ASCII-Wert 32 abziehen.

```
Src_Pointer = &Ascii_1[(b-32)][0];
```

In dem Codeabschnitt ist Ascii_1 das Array welches die Schriftart definiert.

6.6 Ausgeben der Messwerte auf dem Display

Zur Aufbereitung des Messergebnisses wurde die Methode *make_output_string* implementiert, welche das Messergebnis von einem als Ganzzahl gespeicherten Wert in einen zur Augabe geeigneten String überführt. Diese Umwandlung geschieht unter Nutzung der *snprintf*-Methode. Um die Ausgabe auf eine Fixe Länge zu bringen, werden bei Bedarf führende Leerzeichen an den Zahlenwert angehängt. Anschließend werden die Zeichenketten “Distance: “, der Zahlenwert und “cm” aneinandergehägt. Dazu wurde die Methode *str_append* programmiert, welche 2 Zeichenketten aneinander hängt. Die so erhaltene Zeichenkette kann an das Display und an die UART-Schnittstelle zur Ausgabe übergeben werden.

7 Herausforderungen

Als größte Herausforderungen empfanden wir die steile Lernkurve, gerade bei der Ansteuerung des Sensors und der Displays. Beispielsweise benötigten wir sehr viel Einarbeitungszeit um herauszufinden, wie man auf spezifische Register des Sensors zugreift. Das I^2C-Beispielprogramm des Sensors war in Bascom-Basic geschrieben, womit wir wenig anfangen konnten, und auch das Beispiel im SpartanMC Manual gibt diesen Fall nicht vor. Außerdem hat man gerade im SpartanMC Manual das Gefühl, viele unbenötigte Information zu erhalten, die man zur Lösung der Aufgabe nicht benötigt.

Auch die nicht implementierte Verschiebung bei der Nutzung der definierten Schriftart war eine unnötige Herausforderung.